

Chapter 7

Lab 14 Procedures

GOALS

1. Students will be able to create general procedures.
2. Students will be able to create Function Procedures.

ESSENTIAL QUESTIONS

1. Why do you need General Procedures?
2. What is the purpose of Function Procedures?
3. What are the advantages of using General & Function procedures?

LESSON

Visual Basic offers different types of procedures to execute small sections of coding in applications. The various procedures are elucidated in details in this section. Visual Basic programs can be broken into smaller logical components called Procedures. Procedures are useful for condensing repeated operations such as the frequently used calculations, text and control manipulation etc. The benefits of using procedures in programming are:

It is easier to debug a program a program with procedures, which breaks a program into discrete logical limits.

Procedures used in one program can act as building blocks for other programs with slight modifications.

A Procedure can be Sub, Function or Property Procedure.

Call – A general procedure must be called from another procedure in order to be execute. Example:

```
Sub ToBeach()  
    lblDirections.Caption = "Make left at bridge"  
End sub  
Sub Directions()  
    Call ToBeach        'Calls the ToBeach procedure  
End Sub
```

Line Continuation – The character (_) is used for very long lines that goes to the next line.

Example:

```
lblAnswer.Caption = "TextTextTextText TextTextTextText TextTextTextText" _  
    & "More Text"
```

Precondition & Postcondition comments – are comments that tell the user what is true before the procedure is enter and what is true after the procedure executes. **Precondition** – The initial requirements of the procedure before being entered. **Postcondition** – is a statement of what must be true at the end of the procedure execution.

Sub Procedures

A sub procedure can be placed in standard, class and form modules. Each time the procedure is called, the statements between Sub and End Sub are executed. The syntax for a sub procedure is as follows:

```
Sub Procedurename ( )  
    statements  
End Sub
```

arglist is a list of argument names separated by commas. Each argument acts like a variable in the procedure. There are two types of Sub Procedures namely general procedures and event procedures.

Event Procedures

An event procedure is a procedure block that contains the control's actual name, an underscore(_), and the event name. The following syntax represents the event procedure for a Form_Load event.

```
Private Sub Form_Load()  
....statement block..  
End Sub
```

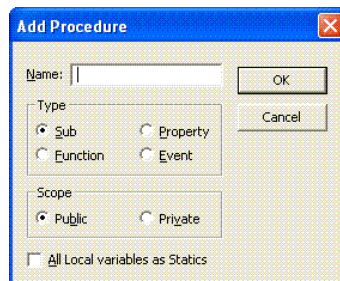
Event Procedures acquire the declarations as Private by default.

General Procedures

A general procedure is declared when several event procedures perform the same actions. It is a good programming practice to write common statements in a separate procedure (general procedure) and then call them in the event procedure.

In order to add General procedure:

- The Code window is opened for the module to which the procedure is to be added.
- The *Add Procedure* option is chosen from the Tools menu, which opens an Add Procedure dialog box as shown in the figure given below.
- The name of the procedure is typed in the Name textbox
- Under *Type*, *Sub* is selected to create a Sub procedure, *Function* to create a Function procedure or *Property* to create a Property procedure.
- Under *Scope*, *Public* is selected to create a procedure that can be invoked outside the module, or *Private* to create a procedure that can be invoked only from within the module.



We can also create a new procedure in the current module by typing Sub ProcedureName, Function ProcedureName, or Property ProcedureName in the Code window. A Function procedure returns a value and a Sub Procedure does not return a value.

Function Procedures

Functions are like sub procedures, except they return a value to the calling procedure. They are especially useful for taking one or more pieces of data, called *arguments* and performing some tasks with them. Then the functions return a value that indicates the results of the tasks complete within the function.

The following function procedure calculates the third side or hypotenuse of a right triangle, where A and B are the other two sides. It takes two arguments A and B (of data type Double) and finally returns the results.

```
Function Hypotenuse (A As Double, B As Double) As Double
Hypotenuse = sqr (A^2 + B^2)
End Function
```

The above function procedure is written in the general declarations section of the Code window. A function can also be written by selecting the *Add Procedure* dialog box from the Tools menu and by choosing the required scope and type.

Property Procedures

A property procedure is used to create and manipulate custom properties. It is used to create read only properties for Forms, Standard modules and Class modules. Visual Basic provides three kind of property procedures-Property Let procedure that sets the value of a property, Property Get procedure that returns the value of a property, and Property Set procedure that sets the references to an object.

PRACTICE

PROJECT 1:

1. Create the following program:

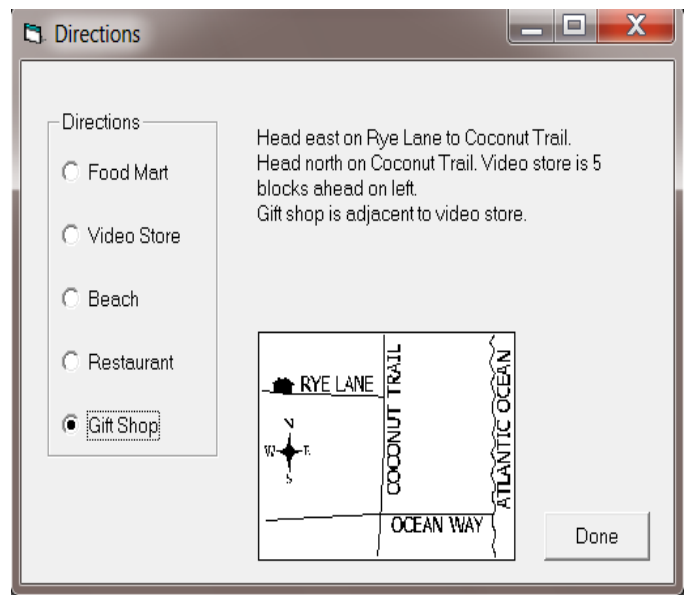
```
' Directions
```

```
Option Explicit
```

```
Private Sub Form_Load()
    optFoodMartDirections.Value = False
    optVideoStoreDirections.Value = False
    optBeachDirections.Value = False
    optRestaurantDirections.Value = False
    optGiftShopDirections.Value = False
End Sub
```

```
Sub ClearDirections()
    lblDirections.Caption = ""
End Sub
```

```
Sub ToCoconutTrail()
    lblDirections.Caption = "Head east on Rye Lane to Coconut Trail." _
        & vbCrLf
End Sub
```



```
Sub ToOceanWay()  
    Call ToCoconutTrail  
    lblDirections.Caption = lblDirections.Caption & _  
        "Head south on Coconut Trail to Ocean Way." & vbCrLf  
End Sub
```

```
Sub ToFoodMart()  
    Call ToOceanWay  
    lblDirections.Caption = lblDirections.Caption & _  
        "Head east on Ocean Way. Food Mart is 3 blocks ahead on right." & vbCrLf  
End Sub
```

```
Sub ToVideoStore()  
    Call ToCoconutTrail  
    lblDirections.Caption = lblDirections.Caption & _  
        "Head north on Coconut Trail. Video store is 5 blocks ahead on left." & vbCrLf  
End Sub
```

```
Sub ToBeach()  
    Call ToOceanWay  
    lblDirections.Caption = lblDirections.Caption & _  
        "Head east on Ocean Way. Beach is straight ahead." & vbCrLf  
End Sub
```

```
Sub ToRestaurant()  
    Call ToBeach  
    lblDirections.Caption = lblDirections.Caption & _  
        "Smitty's is located at the north end of the beach." & vbCrLf  
End Sub
```

```
Sub ToGiftShop()  
    Call ToVideoStore  
    lblDirections.Caption = lblDirections.Caption & _  
        "Gift shop is adjacent to video store." & vbCrLf  
End Sub
```

```
Private Sub optFoodMartDirections_Click()  
    Call ClearDirections  
    Call ToFoodMart  
End Sub
```

```
Private Sub optVideoStoreDirections_Click()  
    Call ClearDirections  
    Call ToVideoStore  
End Sub
```

```
Private Sub optBeachDirections_Click()  
    Call ClearDirections  
    Call ToBeach  
End Sub
```

```
Private Sub optRestaurantDirections_Click()
    Call ClearDirections
    Call ToRestaurant
End Sub
```

```
Private Sub optGiftShopDirections_Click()
    Call ClearDirections
    Call ToGiftShop
End Sub
```

```
Private Sub cmdDone_Click()
    Unload Me
End Sub
```

QUESTIONS

1. What is the purpose of the Call statement?
2. Can you have more than one Call statement in a procedure?
3. Examine the code, what are some advantages of using general sub procedures?
4. What symbol is used for line continuation?

PROJECT 2:

1. Create the following program

```
' Employee Info
Option Explicit

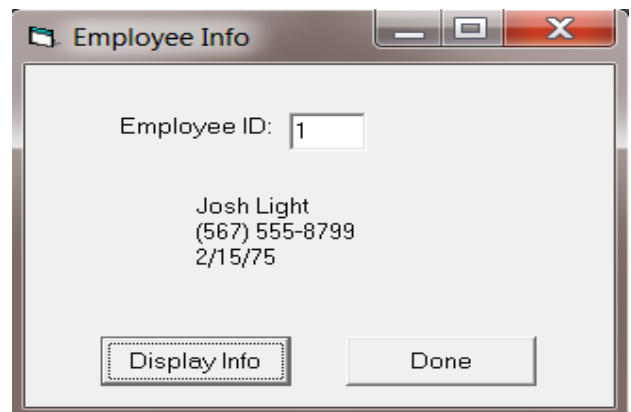
Private Sub Form_Load()

End Sub

Private Sub txtEmployeeID_Change()
    lblInformation.Caption = ""
End Sub

Private Sub cmdDisplayInfo_Click()
    Dim intEmployeeID As Integer
    intEmployeeID = txtEmployeeID.Text

    If intEmployeeID = 1 Then
        Call Employee1
    ElseIf intEmployeeID = 2 Then
        Call Employee2
    ElseIf intEmployeeID = 3 Then
        Call Employee3
    ElseIf intEmployeeID = 4 Then
```



```

        Call Employee4
    Else
        Call Employee5
    End If
End Sub

```

```

Private Sub Employee1()
    lblInformation.Caption = "Josh Light" & vbCrLf & "(567) 555-8799" & _
        vbCrLf & "2/15/75"
End Sub

```

```

Private Sub Employee2()
    lblInformation.Caption = "Kip Pintar" & vbCrLf & "(567) 555-8239" & _
        vbCrLf & "3/17/70"
End Sub

```

```

Private Sub Employee3()
    lblInformation.Caption = "Beverly Hajdu" & vbCrLf & "(567) 555-4949" & _
        vbCrLf & "12/5/65"
End Sub

```

```

Private Sub Employee4()
    lblInformation.Caption = "Juan Cardozo" & vbCrLf & "(567) 555-6537" & _
        vbCrLf & "6/23/45"
End Sub

```

```

Private Sub Employee5()
    lblInformation.Caption = "Brooke Walters" & vbCrLf & "(567) 555-9314" & _
        vbCrLf & "1/1/38"
End Sub

```

```

Private Sub cmdDone_Click()
    Unload Me
End Sub

```

PROJECT 3

1. Create the following program
' Guess Number

Option Explicit

```
Private intSecretNumber As Integer
```

```

Private Sub Form_Load()
    Randomize
    intSecretNumber = Int(100 * Rnd + 1)
End Sub

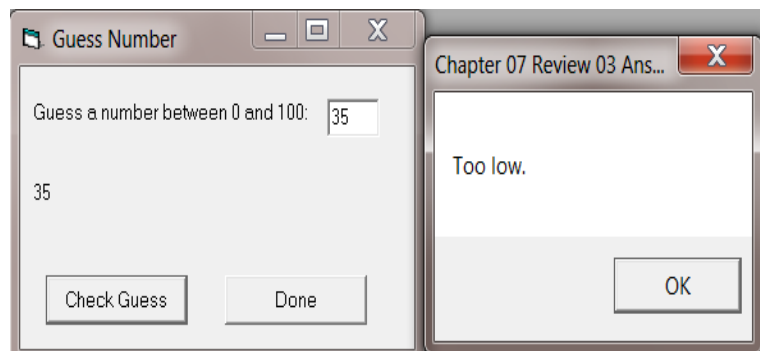
```

```

Private Sub cmdCheckGuess_Click()
    Dim intUserGuess As Integer

    intUserGuess = txtGuess.Text

```



```

' Add guess to label
lblNumbersGuessed.Caption = lblNumbersGuessed.Caption & intUserGuess & " "

If intUserGuess <> intSecretNumber Then
    Call GiveHint(intSecretNumber, intUserGuess)
    txtGuess.Text = ""
Else
    MsgBox "You guessed it!"
End If
End Sub

'*****
' Determine if intSecondNum is larger than intFirstNum and then display an
' appropriate message.
'
' post: A message is displayed in a message box.
'*****
Sub GiveHint(ByVal intFirstNum As Integer, ByVal intSecondNum As Integer)
    If intFirstNum > intSecondNum Then
        MsgBox "Too low."
    Else
        MsgBox "Too high."
    End If
End Sub

Private Sub cmdDone_Click()
    Unload Me
End Sub

```

QUESTIONS:

1. What is the importance of the comment above?
2. Is the comment above a precondition or postcondition comment?
3. What is the difference between a precondition & postcondition comment?

ByRef and ByVal

Parameters can be sent to a subroutine By Reference (ByRef) or By Value (ByVal). ByRef is the default, and means that changes to the variable in the subroutine will result in changes to the source variable outside of the subroutine. ByVal literally copies the values of the variables from the calling subroutine into the called subroutine. By doing this, the variables can be changed, but their values will not change outside of the called subroutine. ByVal can also be a lot slower with large variable types, however, since memory has to be copied from one location to another. If you don't have any reason to do so, there is no need to pass variables ByVal. You can explicitly state the way that a variable is passed to a subroutine by using these keywords before the variable name. Using the ByRef keyword, one could write a Swap function, which switches the values of 2 variables.

```
Private Sub Swap(ByRef x As Integer, ByRef y As Integer)
```

```
    Dim temp As Integer
```

```
    temp = x
```

```
    x = y
```

```
    y = temp
```

```
End Sub
```

```
Private Sub DisplayVals(ByRef a As Integer, ByVal b As Integer)
```

```
    'Don't worry about understanding the next line yet, it will be explained later
```

```
    MsgBox "a = " & CStr(a) & vbCrLf & "b = " & CStr(b)
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    Dim a As Integer
```

```
    Dim b As Integer
```

```
    a = 10
```

```
    b = 12
```

```
    'Display values, swap, and display again
```

```
    DisplayVals a, b
```

```
    'The next line is functionally identical to "Swap a, b"
```

```
    Call Swap(a, b)
```

```
    DisplayVals a, b
```

```
End Sub
```

Notice that Call was used instead of simply stating the subroutine name. When using the Call method however, you must use parenthesis when calling the subroutine. Note that this program would also have worked without typing "ByRef" anywhere, since it is the default.

PROJECT 4

1. Create the following

```
' Restaurant Rating
```

Option Explicit

```
Private Sub cmdOverallRating_Click()
```

```
    Dim intAmbianceRating As Integer
```

```
    Dim intFoodRating As Integer,
```

```
    intServiceRating As Integer
```

```
    Dim intAvgRating As Integer
```

```
    intAmbianceRating = txtAmbiance.Text
```

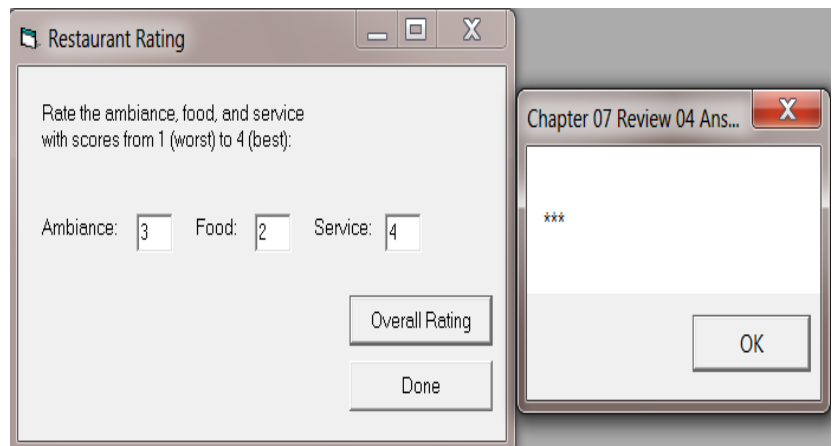
```
    intFoodRating = txtFood.Text
```

```
    intServiceRating = txtService.Text
```

```
    intAvgRating = (intAmbianceRating + intFoodRating + intServiceRating) / 3
```

```
    Call DisplayRating(intAvgRating)
```

```
End Sub
```



```
*****
```

```
' Displays a string of asterisks in a message box
```

```
,
```

```
' post: intScore asterisks displayed in a message box.
```

```
*****
```

```
Sub DisplayRating(ByVal intScore As Integer)
```

```
    MsgBox String(intScore, "*")
```

```
End Sub
```

```
Private Sub cmdDone_Click()
```

```
    Unload Me
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
End Sub
```

QUESTIONS:

1. Why is the argument the variable between the () DisplayRating(*intAvgRating*) important?

2. What is the purpose of “ByVal intScore As Integer” in the code above?

3. What is the difference between ByVal & ByRef?

Lesson Parameters

Parameters, also called Arguments, are variables that can be "passed into" a subroutine. A subroutine with parameters looks like this:

```
Private Sub DisplayAdd(x As Integer, y As Integer)
```

```
    MsgBox x + y
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    DisplayAdd 5, 2
```

```
End Sub
```

A new subroutine has been declared called DisplayAdd. This declaration is different than the declarations that you have seen so far, however, as code has been added between the parenthesis. From your knowledge of variables, this syntax should look somewhat similar to you. x As Integer and y As Integer appear to be variable declarations without the "Dim" keyword. These declarations are separated by a comma. These variables are the Parameters for the DisplayAdd subroutine. Code within the subroutine can access x and y as usual, as if they were normal variables. However, when the subroutine is called, the calling subroutine will also provide values for these parameters. Therefore, the subroutine has now become dynamic. The code can act on input without caring where the input came from. When the Form_Load subroutine calls DisplayAdd with the parameters 5 and 2, the code within DisplayAdd is executed. The first line adds x and y together and displays the result. x and y have already been filled with the values 5 and 2 from the Form_Load subroutine. The calling subroutine doesn't have to use numeric constants, however. It can use variables as well:

```
Private Sub DisplayAdd(x As Integer, y As Integer)
    MsgBox x + y
End Sub
```

```
Private Sub Form_Load()
    Dim a As Integer
    Dim b As Integer

    a = 5
    b = 2
    DisplayAdd a, b
End Sub
```

This code has identical results. Note that DisplayAdd cannot access a and b. As far as DisplayAdd is concerned, a and b are represented as x and y. Attempting to access a or b from DisplayAdd would result in an error.

PROJECT 5

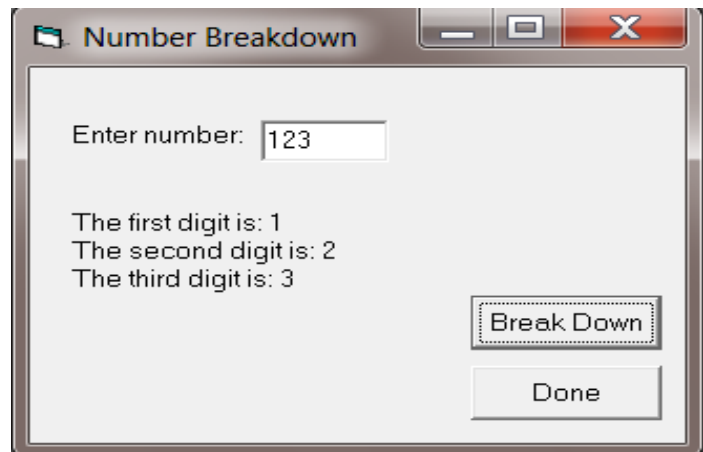
1. Create the following program
' Number Breakdown

Option Explicit

```
Private Sub txtNumber_Change()
    lblNumberBreakdown.Caption = ""
End Sub
```

```
Private Sub cmdBreakDown_Click()
    Dim intNumberEntered As Integer
    Dim intTensDigit As Integer, intOnesDigit As Integer
    Dim intHundredsDigit As Integer
```

```
    intNumberEntered = txtNumber.Text
    If intNumberEntered < 10 Then
        lblNumberBreakdown.Caption = "The first digit is: " & intNumberEntered
    ElseIf intNumberEntered < 100 Then
        Call TwoDigits(intNumberEntered, intTensDigit, intOnesDigit)
        lblNumberBreakdown.Caption = "The first digit is: " & intTensDigit & _
            vbCrLf & "The second digit is: " & intOnesDigit
    ElseIf intNumberEntered < 1000 Then
        Call ThreeDigits(intNumberEntered, intHundredsDigit, intTensDigit, intOnesDigit)
        lblNumberBreakdown.Caption = "The first digit is: " & intHundredsDigit & _
            vbCrLf & "The second digit is: " & intTensDigit & _
            vbCrLf & "The third digit is: " & intOnesDigit
    Else
        lblNumberBreakdown.Caption = "Enter a number less than 1,000."
    End If
End Sub
```



```

*****
' The first digit of intNum is returned as intFirstDigit. The
' second digit of intNum is returned as intSecondDigit.
' pre: intNum is a number less than 100.
' post: intFirstDigit is a number between 0 and 9 inclusive.
' intSecondDigit is a number between 0 and 9 inclusive.
*****
Sub TwoDigits(ByVal intNum As Integer, ByRef intFirstDigit As Integer, ByRef intSecondDigit As Integer)
    intFirstDigit = intNum \ 10
    intSecondDigit = intNum Mod 10
End Sub

*****
' The first digit of intNum is returned as intFirstDigit. The
' second digit of intNum is returned as intSecondDigit. The third
' digit of intNum is returned as intThirdDigit.
'
' pre: intNum is a number less than 1000.
' post: intFirstDigit is a number between 0 and 9 inclusive.
' intSecondDigit is a number between 0 and 9 inclusive.
' intThirdDigit is a number between 0 and 9 inclusive.
*****
Sub ThreeDigits(ByVal intNum As Integer, ByRef intFirstDigit As Integer, _
    ByRef intSecondDigit As Integer, ByRef intThirdDigit As Integer)

    intFirstDigit = intNum \ 100
    intNum = intNum Mod 100
    Call TwoDigits(intNum, intSecondDigit, intThirdDigit)
End Sub

Private Sub cmdDone_Click()
    Unload Me
End Sub

```

QUESTIONS:

1. What symbol is used to extend a statement to the next line?
2. Why did they use ByRef over ByVal?
3. Modify program so that it can handle 4 digits.

PROJECT 6

1. Create the following program
- ```

' Sort Numbers

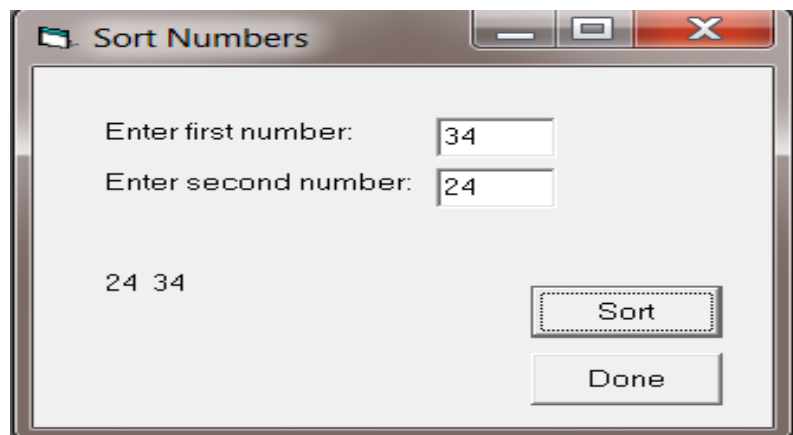
```

Option Explicit

```

Private Sub txtFirstNumber_Change()
 lblAnswer.Caption = ""
End Sub

```



```
Private Sub txtSecondNumber_Change()
```

```
 lblAnswer.Caption = ""
```

```
End Sub
```

```
Private Sub cmdSort_Click()
```

```
 Dim intNum1 As Integer, intNum2 As Integer
```

```
 intNum1 = txtFirstNumber.Text
```

```
 intNum2 = txtSecondNumber.Text
```

```
 Call LowestToHighest(intNum1, intNum2)
```

```
 lblAnswer.Caption = intNum1 & Space(2) & intNum2
```

```
End Sub
```

```

```

```
' Determines if intLowest is actually the lesser of the two values and then
```

```
' swaps intLowest and intHighest if necessary.
```

```
,
```

```
' post: intLowest is assigned the lesser of the two arguments passed.
```

```
' intHighest is assigned the greater of the two arguments passed.
```

```

```

```
Sub LowestToHighest(ByRef intLowest As Integer, ByRef intHighest As Integer)
```

```
 Dim intTemp As Integer
```

```
 If intLowest > intHighest Then ' swap values
```

```
 intTemp = intLowest
```

```
 intLowest = intHighest
```

```
 intHighest = intTemp
```

```
 End If
```

```
End Sub
```

```
Private Sub cmdDone_Click()
```

```
 Unload Me
```

```
End Sub
```

## QUESTIONS:

1. Why was intTemp used to swap values?

2. Modify program to go from Highest to Lowest:

## Statics

Statics allow you to maintain a variables value even when it goes out of scope. For example:

Suppose you have a Sub in a form called CountThem that looks like this:

```
Private Sub CountThem()
```

```
 Dim intI As Integer
 Static intJ As Integer
 intI = intI + 1
 intJ = intJ + 1
 Print intI, intJ
```

```
End Sub
```

Suppose you have some other Sub that calls CountThem three times in a row:

```
Call CountThem
Call CountThem
Call CountThem
```

The following output would be displayed on the form (by default, the "Print" statement directs its output to the current form):

(values for intI)

1  
1  
1

(values for intJ)

1  
2  
3

Note that the "regular" variable, intI, declared with "Dim", does not retain its value between calls, whereas the Static variable, intJ, does.

Note: The keyword "Static" can also be used in the Sub procedure header, which causes all variables in that procedure to be static. Example:

```
Private Static Sub AllVarsAreStatic()
 Dim intCounter As Integer ' as if declared Static
 Dim strErrMsg As String ' as if declared Static
 ... ' other statements
End Sub
```

## PROJECT 7

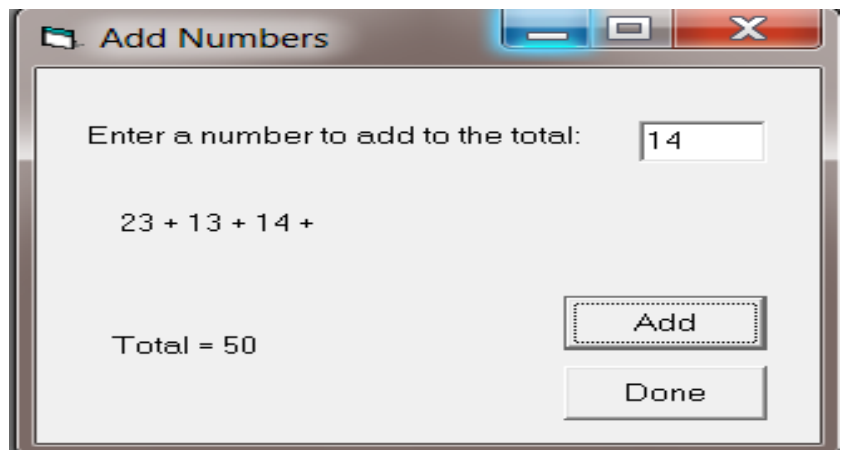
1. Create the following program  
' Add Numbers

Option Explicit

```
Private Sub cmdAdd_Click()
 Dim dblNumberEntered As Double
 Static dblTotal As Double

 dblNumberEntered = txtNewNum.Text
 dblTotal = dblTotal + dblNumberEntered ' running total

 lblNumbersAdded.Caption = lblNumbersAdded & " " _
 & dblNumberEntered & " +"
 lblTotal.Caption = "Total = " & dblTotal
End Sub
```



```
Private Sub cmdDone_Click()
 Unload Me
End Sub
```

## QUESTIONS:

1. What is the purpose of Static?
2. Why not use Static for all variable declarations?

## PROJECT 8

' Letter Grade

Option Explicit

```
Private Sub Form_Load()
```

```
End Sub
```

```
Private Sub txtScore_Change()
```

```
 lblLetterGrade.Caption = ""
```

```
End Sub
```

```
Private Sub cmdLetterGrade_Click()
```

```
 Const dblLowestScore As Double = 0
```

```
 Const dblHighestScore As Double = 100
```

```
 Dim dblScoreEntered As Double
```

```
 dblScoreEntered = txtScore.Text
```

```
 If Not ValidEntry(dblScoreEntered, dblHighestScore, dblLowestScore) Then
```

```
 MsgBox "Enter a score between " & dblLowestScore & " and " & dblHighestScore
```

```
 txtScore.Text = ""
```

```
 lblLetterGrade.Caption = ""
```

```
 Else
```

```
 lblLetterGrade.Caption = "Your grade is " & LetterGrade(dblScoreEntered)
```

```
 End If
```

```
End Sub
```

```

```

```
' Returns True if intLowerLimit < intUserNum < intUpperLimit
```

```
,
```

```
' post: True returned if intLowerLimit < intUserNum < intUpperLimit
```

```
' otherwise False returned
```

```

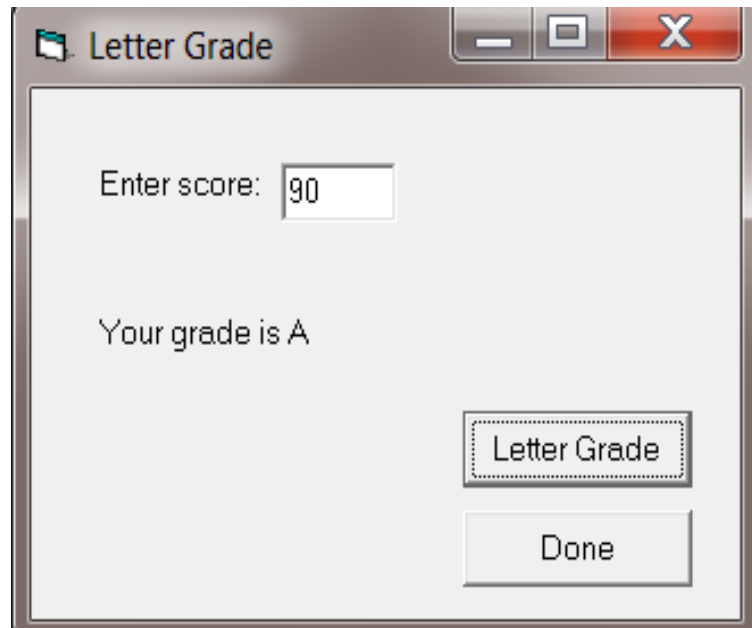
```

```
Function ValidEntry(ByVal intUserNum As Integer, ByVal intUpperLimit As Integer, _
```

```
 ByVal intLowerLimit As Integer) As Boolean
```

```
 If intUserNum > intUpperLimit Or intUserNum < intLowerLimit Then
```

```
 ValidEntry = False
```



```

Else
 ValidEntry = True
End If
End Function

'*****
' Returns a letter grade corresponding to dblScore
',
' post: a letter grade returned
'*****

Function LetterGrade(ByVal dblScore As Double) As String
 If dblScore >= 90 Then
 LetterGrade = "A"
 ElseIf dblScore >= 80 Then
 LetterGrade = "B"
 ElseIf dblScore >= 70 Then
 LetterGrade = "C"
 ElseIf dblScore >= 60 Then
 LetterGrade = "D"
 Else
 LetterGrade = "F"
 End If
End Function

Private Sub cmdDone_Click()
 Unload Me
End Sub

```

## QUESTIONS

1. What is the purpose of the Function procedure?
2. What is the purpose of the data type at the end of a Function?

## PROJECT 9

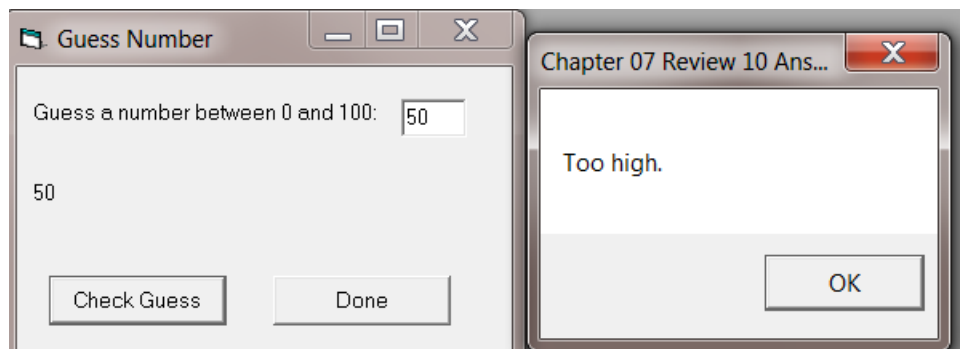
1. Create the following program

```
' Guess Number
```

Option Explicit

```
Private intSecretNumber As Integer
```

```
Private Sub Form_Load()
 Randomize
 intSecretNumber = Int(100 * Rnd + 1)
End Sub
```



```

Private Sub cmdCheckGuess_Click()
 Dim intUserGuess As Integer

 intUserGuess = txtGuess.Text

 ' Add guess to label
 Call DisplayMessage(lblNumbersGuessed, lblNumbersGuessed.Caption & intUserGuess & " ")

 If intUserGuess <> intSecretNumber Then
 Call GiveHint(intSecretNumber, intUserGuess)
 txtGuess.Text = ""
 Else
 MsgBox "You guessed it!"
 End If
End Sub

```

```

'*****
' Determine if intSecondNum is larger than intFirstNum and then display an
' appropriate message.
,

```

```

' post: A message is displayed in a message box.
'*****

```

```

Sub GiveHint(ByVal intFirstNum As Integer, ByVal intSecondNum As Integer)
 If intFirstNum > intSecondNum Then
 MsgBox "Too low."
 Else
 MsgBox "Too high."
 End If
End Sub

```

```

'*****
' Displays a message in a label object
,

```

```

' post: strMessage is assigned to the Caption of lblLabel
'*****

```

```

Sub DisplayMessage(ByRef lblLabel As Label, ByVal strMessage As String)
 lblLabel.Caption = strMessage
End Sub

```

```

Private Sub cmdDone_Click()
 Unload Me
End Sub

```

## PROJECT 10

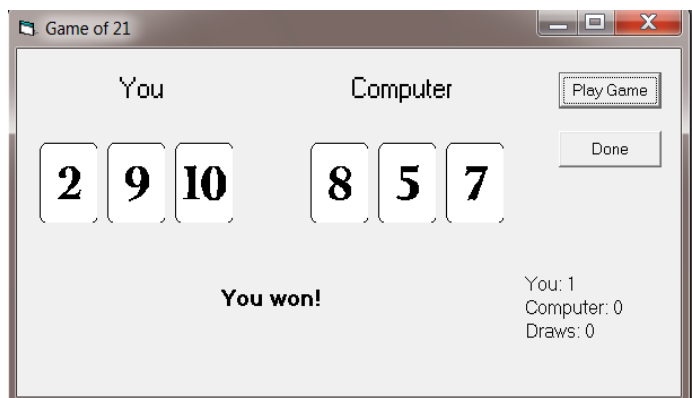
1. Create the following program

Option Explicit

```

Private Const strPlayer As String = "Player"
Private Const strComputer As String = "Computer"

```



```
Private Const strDraw As String = "Draw"
```

```
Private Sub Form_Load()
```

```
 Randomize
```

```
End Sub
```

```
Private Sub cmdPlayGame_Click()
```

```
 Static intPlayerScore As Integer, intCompScore As Integer, intDrawsScore As Integer
```

```
 Dim intPlayerTotal As Integer, intCompTotal As Integer
```

```
 'Deal 3 cards to Player
```

```
 Call DealCard(imgPlayerCard1, intPlayerTotal)
```

```
 Call DealCard(imgPlayerCard2, intPlayerTotal)
```

```
 Call DealCard(imgPlayerCard3, intPlayerTotal)
```

```
 'Deal 3 cards to Computer
```

```
 Call DealCard(imgCompCard1, intCompTotal)
```

```
 Call DealCard(imgCompCard2, intCompTotal)
```

```
 Call DealCard(imgCompCard3, intCompTotal)
```

```
 If Winner(intPlayerTotal, intCompTotal) = strPlayer Then
```

```
 lblWinner.Caption = "You won!"
```

```
 Call UpdateScore(intPlayerScore)
```

```
 Call ShowScore(lblScore, intPlayerScore, intCompScore, intDrawsScore)
```

```
 ElseIf Winner(intPlayerTotal, intCompTotal) = strComputer Then
```

```
 lblWinner.Caption = "Computer won!"
```

```
 Call UpdateScore(intCompScore)
```

```
 Call ShowScore(lblScore, intPlayerScore, intCompScore, intDrawsScore)
```

```
 Else
```

```
 lblWinner.Caption = "It's a draw!"
```

```
 Call UpdateScore(intDrawsScore)
```

```
 Call ShowScore(lblScore, intPlayerScore, intCompScore, intDrawsScore)
```

```
 End If
```

```
End Sub
```

```

```

```
' Displays a card corresponding to a random number in the range 1 to 10
```

```
,
```

```
' pre: Randomize called
```

```
' post: imgCard displays a card in the range 1 to 10 and updates intPlayerTotal score
```

```

```

```
Sub DealCard(ByRef imgCard As Image, ByRef intPlayerTotal As Integer)
```

```
 Dim intCardNum As Integer
```

```
 intCardNum = Int(10 * Rnd + 1)
```

```
 If intCardNum = 1 Then
```

```
 imgCard.Picture = LoadPicture("Card1.wmf")
```

```
 ElseIf intCardNum = 2 Then
```

```
 imgCard.Picture = LoadPicture("Card2.wmf")
```

```
 ElseIf intCardNum = 3 Then
```

```
 imgCard.Picture = LoadPicture("Card3.wmf")
```

```
 ElseIf intCardNum = 4 Then
```

```

 imgCard.Picture = LoadPicture("Card4.wmf")
 ElseIf intCardNum = 5 Then
 imgCard.Picture = LoadPicture("Card5.wmf")
 ElseIf intCardNum = 6 Then
 imgCard.Picture = LoadPicture("Card6.wmf")
 ElseIf intCardNum = 7 Then
 imgCard.Picture = LoadPicture("Card7.wmf")
 ElseIf intCardNum = 8 Then
 imgCard.Picture = LoadPicture("Card8.wmf")
 ElseIf intCardNum = 9 Then
 imgCard.Picture = LoadPicture("Card9.wmf")
 ElseIf intCardNum = 10 Then
 imgCard.Picture = LoadPicture("Card10.wmf")
 End If
 intPlayerTotal = intPlayerTotal + intCardNum
End Sub

```

```

```

```

' Returns a string indicating the winner

```

```

,

```

```

' post: global constant strPlayer, strComputer, or strDraw returned

```

```

```

```

Function Winner(ByVal intPlayerTotal As Integer, ByVal intCompTotal As Integer) As String
 Const intLimit As Integer = 21

```

```

 If (intPlayerTotal = intCompTotal) Or _
 (intPlayerTotal > intLimit And intCompTotal > intLimit) Then
 Winner = strDraw
 ElseIf (intCompTotal > intLimit) Or _
 (intPlayerTotal > intCompTotal And intPlayerTotal <= intLimit) Then
 Winner = strPlayer
 Else
 Winner = strComputer
 End If
End Function

```

```

```

```

' Updates the score of intWinner

```

```

,

```

```

' post: intWinner has been increased by intWinPoints

```

```

```

```

Sub UpdateScore(ByRef intWinner As Integer)
 Const intWinPoints As Integer = 1

```

```

 intWinner = intWinner + intWinPoints
End Sub

```

```

```

```

' Displays the current score of Player, Computer, and draws

```

```

,

```

```

' post: scores displayed in a label

```

```

```

```
Sub ShowScore(ByRef lblLabel As Label, ByVal intPlayerScore As Integer, _
ByVal intCompScore As Integer, ByVal intDrawsScore As Integer)
 lblScore.Caption = "You: " & intPlayerScore & vbCrLf _
 & "Computer: " & intCompScore & vbCrLf _
 & "Draws: " & intDrawsScore
End Sub
```

```
Private Sub cmdDone_Click()
 Unload Me
End Sub
```